CISCO SYSTEMS

Document Number	EDCS-194266
Revision	0.8
Author	Doug Wooff, Carol Gal
Project Manager	Mala Devlin
Commit Status	Not Committed

HFR Boot Architecture and Optimizations Functional Requirement Document

Project Headline

This document defines product requirements for the HFR boot architecture and optimizations.

Department	Name and Title	Approval Date
	Mala Devlin, Manager, DSM Software	
	Ashok Ganesan, HFR Product Manager	
	David Ward, HFR Software Architect	
	Ayman Sayed, Director, HFR Software Development	

Modification History

Rev	Date	Originator	Comment
0.1	[redacted]	Carol Gal, Doug Wooff	alpha draft
0.2	[redacted]	Carol Gal, Doug Wooff	add more content
0.3	[redacted]	Carol Gal, Doug Wooff	update content
0.4	[redacted]	Carol Gal, Doug Wooff	add 'Scenarios and Dependencies'
0.5	[redacted]	Carol Gal, Doug Wooff	fill in sections 5.X
0.6	[redacted]	Carol Gal, Doug Wooff	made corrections on 5.X for dSC election algorithm, added sysmgr requirements
0.7	[redacted]	Carol Gal, Doug Wooff	update after review
0.8	[redacted]	Carol Gal, Doug Wooff	add requirement for rommon reading MBI files, final review updates

EXHIBIT 1.4

1.0 Overview

This document contains a list of boot architecture and optimization Requirements for HFR. It is intended as an auxiliary document to the overall HFR Product Requirements Document ENG-47829.

This document refers only to the case of booting HFR entities, such as racks and Logical Routers, with a composite software package configuration as it will be delivered in QFT3. It lists the feature requirements of the system and the components involved.

2.0 Definitions

Requirements are identified using the following designation (with nnn corresponding to a sequential number):

- R-nnn - Mandatory requirement. Initial release of the product (FCS) can not be shipped without this requirement being met.
- F-nnn - Future requirement. This requirement has to be met in the future software releases of the product (post FCS). Note that the FCS and post-FCS phasing is with respect to software requirements only, the hardware needs to support both R-nnn and FR-nnn features from start.
- O-nnn - Objective. A desirable characteristic that can enhance the product but may be omitted if required by timing, resource or other constraints. Hardware support for O-nnn requirements is optional, *i.e.*, if can be easily provided without seriously affecting performance, schedule, *etc.*

3.0 System Requirements for Boot Scenarios

3.1 Initial (Cold) Boot

R-3.1.1 17 minute absolute maximum rack cold boot time.

The first poweron of a newly procured HFR rack will, in general, require user attention. Every type of HFR rack requires at least one SC/SCRP with "installed" software to perform any useful work. Some default software release is shipped with every SC/SCRP card, "burned in" to its internal flash storage. This allows a user to bring up a rack, insert cards, check the inventory, assign cards to LR's, and configure them for use.

Initial cold boot time of a rack must not be "excessive", but it will take some time and requires user intervention. Consider that the rack can do no meaningful work until configured by the user. In addition, it is likely that a user will populate a rack first with an SC/SCRP, then carefully insert LC's and DRP's as the SCRP boots its default software. A rack is not shipped with cards pre-inserted.

Whether a user populates all the cards and powers on a rack, or whether a staged insertion of cards is performed, time is required to let the cards boot, aquire and cache software, and accept configurations.

Configuration files can be sourced from a nearby file server where it has been pre-edited. It is in this case that the maximum rack cold boot time will become most meaningful.

At any time, the user can bring in a fresh install disk, supplied by Cisco, to override the default factory-shipped software. This will require additional time to redistribute and reconfigure the new software.

As cards are inserted into racks, they will pull in software assigned to them, and it gets stored (ie. cached) locally on all classes of cards.

A figure of 10 minutes is proposed as a ***target*** time for the factory-provided software to boot and self-distribute on whatever cards are present. This assumes that there will exist some rules on what to distribute without any LR created by a user. Certain default packages may self-distribute within the owner LR, for example.

The current plan is to boot all discovered cards, even if no config is supplied to direct cards to an LR. The card itself will not be shutdown, but any physical interfaces will be shutdown in the absence of any config. Cards which are not explicitly configured into an LR will be placed in LR0 (LR-zero), and loaded with default software.

R-3.1.2 Define rules for initial default package distribution.

3.2 System Boot

R-3.2.1 5 minute maximum system reboot time.

A system boot, either single- or multi-rack, subsequent to the initial bringup (after a power outage, for example), will be faster, and must restore the ***system*** to working order within 5 minutes. We can note that this is a ***best-case*** scenario, wherein no flash writes are needed, and no user interaction is required. All software and configurations are already distributed and cached.

In this best-case scenario, the maximum time includes configurations being applied, interfaces coming up, but route convergence and hence packet forwarding may not yet have been achieved. Note that, for a multi-rack system, all fabric racks must come up during this time.

After an office-wide power outage, all racks will boot in parallel. To achieve 5 minute system boot time, each rack must be "ready" somewhat sooner than 5 minutes.

It is envisioned that each CPU node will require its software plus all configurations to be already cached in local (flash) storage in order to achieve acceptable rack boot time.

The user need only configure the system once. After that, the rack boot time requirement applies.

3.3 LC OIR Boot Time

R-3.3.1 The best-case LC reload time, after initial confiurations are already in place, must be 30 seconds or less.

4.0 Feature Requirements List by Functional Components

4.1 Distributed Software Management (DSM) Boot Services

R-4.1.1 DSM will provide management of software configurations for different card-types/nodes/ LRs. This includes the management of multiple install streams for a node in the system, the activation of a specific install stream for any given node and package upgrade and downgrade scenarios

R-4.1.2 DSM will provide an MBI (ie. OS package) selection mechanism for a node based on its RSM address, card type and LR membership.

R-4.1.3 DSM will always attempt to use locally cached versions of MBI and packages on each node. Whenever they are not available, upon downloading them from the boot server source, a local copy on the node's persistent storage device will be made to speedup subsequent boot events

O-4.1.4 DSM should make use of software distribution efficiencies whenever possible during the boot process in order the improve the boot time performance. An analysis will be made on the performance of the existing design on HFR hardware. Enhancements will be recommended as required.

4.2 System Manager Process (SysMgr)

R-4.2.1 SysMgr will map the startup levels of all startup processes into no more than three distinct categories, also known as "bands". This mapping will be employed through the use of a Master File Map that will allow the flexibility of changing the boundaries of the levels within each category

R-4.2.2 SysMgr will block after the completion of each category, waiting for acknowledgement from all the processes before proceeding to the next band.

R-4.2.3 All the processes belonging to the same category will be spawned by SysMgr in the ascending order of their startup level (natural order), unless otherwise specified. SysMgr will not block waiting for acknowledgement while spawning processes within the same band.

R-4.2.4 As a direct result from R-4.2.3, all processes that are spawn by SysMgr should not make assumptions on existing dependencies on each other; i.e. if a newly spawned process depends on the services of another process and they are not available, it should provide a delay and check the service again, thus relinquishing the cpu usage.

R-4.2.5 The Master File Map should support a keyword for changing the natural startup order (see R-4.2.3). At least two different startup orders should be provided: *normal* order - ascending order of the startup levels (default) and *reverse* order - descending order of the startup levels. This feature will be used only internally during testing for compliance with R-4.2.4.

External Requirement: All processes started by sysmgr should be capable of coming up in any relative order, without assuming that any services are already present at the time of their spawning. If this requirement is found to be broken during internal tests using any type of spawning order, ddts's will be raised against those components.

R-4.2.6 SysMgr will provide a way to reproduce the startup order for each particular boot scenario, including the reverse order.

R-4.2.7 An enhanced version of 'init', called 'sysmgr-lite', is required in order to simplify and consolidate sysmgr functionality within the MBI. Sysmgr-lite will be essentially a fault-tolerant version of 'init' that will support process restart and a minimal set of sysmgr api's.

Sysmgr-lite will cater only to "local" processes, i.e. those processes already present within the monolithic MBI image. Sysmgr-lite will not need to concern itself with any processes pulled down in new or upgraded packages. Sysmgr-lite will hand over control of current processes to "sysmgr-full" (aka 'sysmgr') as soon as the first new package is pulled (method TBD). Sysmgr-full will handle more complex process attributes such as tuple triggering, placeability, etc. A "handoff" mechanism must be designed such that sysmgr can take over from sysmgr-lite after the MBI pulls and restarts packages.

R-4.2.8 Sysmgr will handle all new packages and support an infinite number of package upgrades or downgrades.

R-4.2.9 The design will allow for straightforward and seamless upgrades of sysmgr itself. Having 'shadow' versions of sysmgr (and possibly sysmgr-lite) will continue to be the preferred method of dealing with a failure of sysmgr itself.

4.3 MBI Evolution

The basic premise when creating a platform MBI is to keep it small and simple. Upgrades to components within the MBI, ie. within the OS package, are inherently more difficult and more impactful.

R-4.3.1 Remove QSM from OS package: all CPU targets

It was previously not possible to remove QSM from the MBI (ie. from the OS package). It should now be possible on the existing GSR implementation due to the removal of a hard sysmgr dependency on QSM. There are no users of QSM in the MBI. QSM will likely be moved to the Base package.

R-4.3.2 Remove Driver Infra Partner and netio from OS package: PPC targets

It is not possible to remove DiPartner and netio from the GSR OS package without significant redesign efforts. These components can be moved out of the PPC OS package. In the PPC MBI, qnet has no dependencies on DiPartner and netio. On GSR, they are needed for qnet transport.

R-4.3.3 Update mbi-hello process in the OS package: PPC targets

mbi-hello behaves differently in the QFT3 universe. A new version must be created.

R-4.3.4 Ensure Shelf Manager and dSCp cooperate with MBI: PPC targets

The interaction of Shelf Manager, within a rack, with all cards and their MBI's should be reviewed. A complete walk-thought of boot scenarios should be undertaken. Experience gained from QFT2 package deployment may be an asset.

The same review of the interaction of the designated SC process (dSCp) and its "subordinate" SC's should also be scheduled.

Both Shelf Manager and dSCp use MBI Manager as a gateway to Install Manager and LRd. MBI Manager must be created and should be re-reviewed.

4.4 User Interface

Testers must be allowed to run diagnostic images in place of production code.

R-4.4.1 Allow diagnostic images to be run from the CLI.

4.5 Saving MBI images and software packages to disk

R-4.5.1 A copy of the booted MBI image file itself needs to be cached in local flash storage, wherever hardware support is provided for such persistent storage. The MBI image file must be written in such a way that ROMMON can read it at a later time.

R-4.5.2 ROMMON must be able to search for, read and load an MBI image file which was previously written for it by Install Helper.

R-4.5.3 Install Helper on the MBI must cache downloaded packages in local flash storage, wherever hardware support is provided for such persistent storage. This is envisioned to reduce node reboot times.

R-4.5.4 Line card software packages, two versions of the MBI plus local configuration files must fit into the 64M of bootflash (see Note below this paragraph)

R-4.5.5 Flash file system and drivers must provide acceptable burn times

Local MBI and software package caching must be performed either

- during initial software configuration if flash write speed is adequate OR

- deferred until a later time if system boot time is impaired such that requirement R-3.2.1 is not met.

Note:

- there are currently four HFR card classes: RP (ie. SC/SCRP), DRP, LC, and SP.
- LC and SP class of cards will use the bootflash as a persistent storage device to save and cache the active packages.
- (D)RP cards use PCMCIA flash disks (internal or external) as a persistent storage device.

4.6 Performance

4.6.1 SMP impact

The (D)RP nodes in HFR have a dual processor engine in a Symmetric Multi-Processor (SMP) configuration. The CPU is a Motorola Vger 7455 and has three level cache hierarchies that are kept in sync through a cache coherency mechanism. This architecture provides not only more CPU resources but requires a new software design approach to take advantage of the concurrency offered by the two processors. One example is the new System Manager design for HFR (see R-4.2.2) that allows processes to be spawned in parallel without blocking, thus putting both processors to work.

We have developed tools to monitor the CPU usage (kernel, user, idle) and continous efforts are aimed at minimizing the idle time, thus increasing boot performance. Also, at boot time - while in rommon - the development community can specify the mode in which to run, single processor or SMP. We use this feature to measure the speedup factor, defined as the amount of time to perform a given activity (booting in our case) in an SMP configuration divided into the amount of time for the same activity to be completed in a single processor configuration. The result is expressed in percent, and obviously the theoretical limit is 50% if the same activity completes twice as fast in an SMP configuration. Currently, we have increased the speedup factor on the PPC platform during boot from 10% to 25%. There are scripts to monitor this performance indicator as well as absolute boot times, cpu usage on nightly builds: http://q-bert.cisco.com/index.php.

The speedup factor is a very useful indicator that can be applied to other activities as well, protocol convergence, infrastructure performance, etc.

4.6.2 Push vs. Pull mechanism

Current DSM design uses a 'pull' model for each node (ie. each HFR-OS CPU) to fetch the file subset of a package which applies to itself.

The inherent simplicity and node independence of this approach has been demonstrated. However, there may be operational inefficiencies associated with the design. The initial implementation, which performs a file-by-file copy via the remote filesystem, has been found to consume excessive kernel overhead.

If any prevalent inefficiencies cannot be directly addressed, a push model or even a combined pull-push model may be considered.

4.6.3 Multicast

If a 'push' model of software package delivery becomes necessary or advantageous, then multicasting identical software packages to recipient nodes becomes viable. This may be useful to reduce initial boot time, when nodes generally do not have the correct software cached locally. Multicasting software packages could also be useful after a software upgrade in which a particular common package affects many nodes.

In general, it is possible that each HFR node may utilize a different set of software packages. The problem of aligning a set of recipient nodes for receiving multicasted packages may introduce undesired complexity in the design.

Performance of multiple nodes pulling packages individually, with appropriate optimizations, will be evaluated before updating the 'pull' design.

The feasability of such a design change as well as the benefits in terms of performance and the outcome of it will drive this requirement for FCS timeframe.

Note: today's boot mechanism successfully implements a pull mechanism in which every node gets its non-cached packages on an individual base from the boot serve;, there is no multicast involved. Multicasting may yield performance gains if the number of package reuse among the nodes is significant - say as a result of a major upgrade and having many similar nodes within an LR within a rack.

4.6.4 Pre-bundling of Packages

An efficiency improvement being explored is the concept of pre-bundling packages on the RP, ie. on the code server, before allowing the remote node to 'pull' it down via qnet.

It was observed that file-by-file copying of packages incurs tremendous overhead for remote file open, stat, and read operations.

The prototype prebundler creates a thread for each requesting node which fills a large buffer with many files, thereby avoiding much overhead.

Timing results will be used to assess the effectiveness of this design approach. These results can then be used to influence whether additional design changes should be undertaken to allow multicasting of packages to remote nodes.

5.0 References

1. HFR DSM Boot Services, EDCS-131905
2. HFR Distributed Software Management, ENG-150583
3. HFR DSM QFT2 Functional and Design Specification, ENG- 126996
4. HFR MBI Manager Functional Specification, ENG-136393
5. HFR LR Daemon Functional Specification, ENG-108732
6. HFR Designated Shelf Controller dSC Functional Specification, ENG-114800
7. HFR Designated Shelf Controller Design Specification, ENG-155856